Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Interactive 3D Visualization of Data Structures and Algorithms Using AST-Based Trace Generation

Dr. S. B. Chaudhari
Project Guide Computer Department JSPM's JSCOE
Pune, India

Sahil Chand Shaikh
Computer Department
JSPM's JSCOE
Pune, India

Pathan Aamer Khan
Computer Department
JSPM's JSCOE
Pune, India

Inamdar Amaan Ashpak
Computer Department JSPM's JSCOE
Pune, India

Kurmi Yashraj Dayanath
Computer Department JSPM's JSCOE
Pune, India

***Abstract*—Data Structures and Algorithms (DSA) are among the most important subjects in computer science because they help students develop logical thinking and efficient problem-solving skills. However, many learners find it difficult to understand how data structures behave during program execution. Concepts such as tree traversal, graph exploration, linked list manipulation, and sorting operations involve multiple state changes that are often hard to follow using traditional teaching methods. Static diagrams, textbook examples, and console outputs provide limited insight into the actual execution flow of an algorithm, making it challenging for students to connect theory with implementation.**

To address this issue, this paper presents the *DSA Logic Visualizer*, an interactive web-based platform designed to visualize the execution of user-written JavaScript programs in a three-dimensional environment. The platform enables learners to write code, execute it securely, and observe the behavior of various data structures and algorithms through step-by-step animations. The proposed system is built using a Next.js frontend, a Node.js backend, and a visualization layer developed with React Three Fiber and Three.js. A key feature of the platform is the use of Abstract Syntax Tree (AST) transformation, which automatically inserts tracing instructions into user-submitted code. This allows runtime information to be collected without requiring users to manually modify their programs.

The generated execution traces are processed and transformed into interactive visual representations of Arrays, Linked Lists, Trees, Graphs, Stacks, Queues, and common Sorting and Searching Algorithms. The platform also includes playback controls that allow users to pause, replay, and navigate through execution steps at different speeds, making algorithm analysis more accessible and engaging. To ensure safe code execution, the system employs a sandboxed runtime environment with timeout restrictions and a loop protection mechanism that prevents infinite execution.

The proposed solution aims to improve the learning experience of DSA by combining program analysis, secure code execution, and interactive visualization within a single platform. By providing a clear view of how algorithms operate internally, the system helps learners better understand complex concepts, identify logical errors, and strengthen their programming skills. The platform can serve as a valuable educational tool for students, instructors, and beginners seeking a more intuitive approach to learning data structures and algorithms.

***Index Terms*—Data Structures and Algorithms, Algorithm Visualization, Abstract Syntax Tree, Three.js, React Three Fiber, Computer Science Education, Interactive Learning, Sandboxed Execution, Next.js, Node.js**

I. Introduction

Data Structures and Algorithms (DSA) play a vital role in computer science education and software development. They provide the foundation for designing efficient programs and solving complex computational problems. Consequently, DSA is considered one of the most important subjects in undergraduate computer engineering and computer science curricula. Students are expected to understand not only the theoretical concepts behind different data structures but also how algorithms manipulate them during execution. However, learning these concepts is often challenging because many operations occur internally and are not directly visible to the learner.

Traditional methods of teaching DSA typically rely on classroom lectures, textbook examples, static diagrams, and console-based outputs. While these approaches are effective for introducing basic concepts, they often fail to demonstrate how data structures change dynamically as a program executes. Understanding how nodes are connected in a linked list, how a tree grows during insertion, or how graph traversal algorithms explore vertices requires learners to mentally track multiple state changes simultaneously. This process becomes difficult, particularly for beginners who are still developing their programming and problem-solving skills.

Over the years, several visualization tools have been developed to improve the learning experience of DSA. Most existing solutions provide animations for predefined algorithms and allow students to observe how a particular operation works. Although such tools are useful for conceptual understanding, they are generally limited to a fixed set of examples. Students cannot easily visualize the behavior of their own implementations, making it difficult to connect theoretical knowledge with practical coding experience. Similarly, code execution tracing tools provide detailed execution information but often present it in the form of tables or memory diagrams, which

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

may not effectively represent the spatial structure of complex data structures such as trees and graphs.

To overcome these limitations, this work presents the *DSA Logic Visualizer*, an interactive web-based platform that enables users to visualize the execution of their own JavaScript programs through real-time three-dimensional animations. The system combines code execution, automated trace generation, and interactive visualization within a single environment. Users can write or paste code into an integrated editor and observe how different data structures and algorithms evolve step by step during execution. This approach helps learners understand the relationship between source code and runtime behavior more effectively than traditional learning methods.

A key feature of the proposed system is the use of Abstract Syntax Tree (AST) transformation for automatic trace injection. Instead of requiring users to manually insert debugging statements, the platform analyzes the submitted code and automatically generates execution traces. These traces are then converted into visual events that drive the three-dimensional animation engine. The platform supports visualization of Arrays, Linked Lists, Trees, Graphs, Stacks, Queues, and common Sorting and Searching Algorithms, making it suitable for a wide range of DSA topics.

The system is developed using modern web technologies including Next.js, Node.js, React Three Fiber, and Three.js. To ensure secure execution of user-submitted code, a sandboxed runtime environment is employed along with timeout restrictions and loop protection mechanisms. This allows users to experiment freely without compromising system stability or security.

The major contributions of this work are summarized as follows:

- Development of a web-based platform for interactive visualization of Data Structures and Algorithms.
- Implementation of an AST-based automatic trace generation mechanism for user-written JavaScript code.
- Integration of a secure sandboxed execution environment with timeout and loop protection features.
- Design of a three-dimensional visualization engine capable of representing multiple data structures and algorithm categories.
- Provision of playback controls that allow users to analyze program execution step by step.
- Evaluation of the platform's effectiveness in improving algorithm comprehension and learning engagement among students.

By combining program analysis, secure execution, and interactive visualization, the proposed platform aims to make DSA learning more intuitive, engaging, and accessible for students. The system encourages active learning by allowing users to directly observe the effects of their code, thereby reducing the gap between theoretical understanding and practical implementation.

II. Related Work

The use of visualization techniques for teaching Data Structures and Algorithms (DSA) has been widely explored in computer science education. Researchers and educators have developed various tools to help learners understand algorithm execution through graphical representations and interactive demonstrations. These tools have shown that visual learning can improve comprehension, reduce confusion, and increase student engagement. However, many existing solutions have limitations related to flexibility, user interaction, or support for custom code execution.

A. Algorithm Visualization Platforms

Several visualization platforms have been developed to demonstrate the working of common data structures and algorithms. One of the most popular examples is VisuAlgo [1], which provides interactive animations for topics such as sorting, searching, graph algorithms, trees, and heaps. Similarly, Data Structure Visualizations developed by David Galles [3] offers a collection of educational visualizations for various DSA concepts.

Although these platforms are highly useful for learning standard algorithms, they are primarily limited to predefined examples. Users can interact with the visualizations but cannot directly execute and visualize their own implementations. As a result, learners often face difficulties when transitioning from observing demonstrations to writing and debugging their own code.

B. Program Execution Tracing Tools

Another category of educational tools focuses on program execution tracing. Systems such as Python Tutor [2] provide detailed visual representations of program execution by displaying variable values, function calls, and memory states during runtime. These tools help learners understand how code executes internally and are particularly effective for debugging and understanding program logic.

Despite their advantages, execution tracing tools generally represent information using tables, memory diagrams, or textual views. While these approaches effectively display runtime information, they do not always provide intuitive representations of complex structures such as trees, graphs, and linked lists. Furthermore, the visualization experience is often limited to two-dimensional layouts.

C. Interactive Learning Through Visualization

Previous studies in computer science education have reported that students learn more effectively when they actively interact with educational tools rather than passively observing demonstrations. Interactive visualization allows learners to experiment with different inputs, observe the effects of code modifications, and develop a deeper understanding of algorithm behavior. Research findings indicate that active engagement can improve concept retention, problem-solving ability, and overall learning outcomes [6], [13], [14].

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Inspired by these findings, modern educational tools increasingly focus on providing interactive environments where learners can explore concepts independently. However, many existing systems still rely on fixed examples and offer limited support for visualizing user-generated code.

D. Three-Dimensional Visualization in Educational Systems

Three-dimensional visualization has been successfully applied in domains such as medical education, engineering design, scientific simulations, and virtual laboratories. By providing depth and spatial representation, three-dimensional environments help users better understand relationships between objects and processes [15].

In the context of DSA education, however, the use of three-dimensional visualization remains relatively limited. Most available tools continue to rely on traditional two-dimensional interfaces. This creates an opportunity to explore how three-dimensional environments can improve the understanding of hierarchical and interconnected data structures such as trees and graphs while also making algorithm learning more engaging for students.

E. Research Gap

Based on existing literature and currently available educational tools, several limitations can be identified. Many visualization systems focus only on predefined algorithms, while execution tracing tools often lack intuitive graphical representations. Additionally, support for real-time visualization of user-written code combined with interactive three-dimensional rendering remains limited.

The proposed DSA Logic Visualizer addresses these challenges by integrating automatic code tracing, secure sandboxed execution, and interactive three-dimensional visualization within a single platform. Unlike conventional visualizers, the system allows users to execute their own JavaScript programs and observe the behavior of various data structures and algorithms through animated visual representations. This combination of code execution and visualization helps bridge the gap between theoretical learning and practical implementation, providing a more engaging and effective learning experience.

III. Methodology

The DSA Logic Visualizer is designed as a web-based educational platform that converts user-written JavaScript code into interactive three-dimensional visualizations. The system combines source code analysis, runtime tracing, secure execution, and graphical rendering to help learners understand how data structures and algorithms behave during execution. The complete workflow consists of code submission, AST transformation, sandboxed execution, trace generation, and visualization rendering.

A. System Architecture

The proposed platform follows a three-layer architecture consisting of the Frontend Client, Backend Processing Server, and 3D Visualization Engine. Each layer performs a specific responsibility within the execution pipeline.

The frontend allows users to write, edit, and execute JavaScript programs using an integrated code editor. The backend receives the submitted code, performs AST-based trace injection, executes the transformed code inside a secure sandbox, and generates runtime traces. These traces are then transmitted to the visualization engine, which renders the execution process through interactive three-dimensional animations.

The modular design ensures that code processing, execution, and visualization remain independent, making the platform easier to maintain and extend in the future.

B. AST-Based Trace Injection

One of the core features of the proposed system is the automatic generation of execution traces through Abstract Syntax Tree (AST) transformation. When a user submits code, the AST Transformer parses the source program and identifies important programming constructs such as:

- Variable declarations
- Assignment statements
- Function calls
- Loop structures
- Conditional statements
- Data structure operations

After identifying these constructs, the system automatically inserts trace statements into the transformed code. These statements capture runtime information such as variable values, execution states, and structural changes occurring within data structures.

If a program consists of statements

$$P = \{s_1, s_2, s_3, \ldots, s_n\}$$

the transformed program can be represented as

$$P' = \{s_1, \tau_1, s_2, \tau_2, \ldots, s_n, \tau_n\}$$

where τ represents the automatically injected trace operation associated with a program statement.

This approach eliminates the need for manual debugging statements and allows arbitrary user-written programs to be visualized.

C. Sandboxed Execution and Loop Protection

After instrumentation, the transformed code is executed inside a secure sandbox environment implemented using Node.js Virtual Machine contexts. The sandbox restricts access to external resources and prevents potentially harmful operations.

To improve security and reliability, the following restrictions are enforced:

- File system access is disabled.
- Network communication is blocked.
- External module imports are restricted.
- Execution timeout limits are applied.
- Resource usage is controlled.

A Loop Guard mechanism is also implemented to prevent infinite loops. During AST traversal, iteration counters are

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

inserted into loop constructs. If the iteration count exceeds a predefined threshold, execution is terminated automatically and an error message is returned to the user.

This mechanism protects the platform from resource exhaustion while maintaining a smooth user experience.

D. Visualization Mapping

Once execution traces are generated, the visualization engine determines which graphical component should represent the captured runtime state. Each supported data structure is associated with a dedicated visualizer component.

TABLE I
DATA STRUCTURE TO VISUALIZER MAPPING

Data Structure / Algorithm	Visualizer Component	3D Representation
Array	ArrayVisualizer	Ordered cuboid sequence
Linked List	LinkedListVisualizer	Connected nodes with directional links
Binary Tree	TreeVisualizer	Hierarchical node structure
Graph	GraphVisualizer	Connected vertex-edge network
Stack	StackVisualizer	Vertical LIFO arrangement
Queue	QueueVisualizer	Horizontal FIFO arrangement
Sorting Algorithms	AlgoVisualizer	Animated element comparisons
Searching Algorithms	AlgoVisualizer	Stepwise traversal visualization

This mapping layer converts raw execution traces into meaningful visual events that can be easily interpreted by learners.

E. Playback and Animation Engine

The generated execution traces are stored as an ordered sequence of snapshots. The playback engine maintains a cursor that points to the currently displayed snapshot.

Users are provided with the following controls:

- Play
- Pause
- Next Step
- Previous Step
- Speed Adjustment

Whenever the cursor position changes, the corresponding visualization state is rendered. Smooth transitions between states are achieved using spring-based interpolation techniques, allowing movement and color changes to appear natural and easy to follow.

This feature enables learners to revisit specific execution steps and carefully analyze algorithm behavior.

F. System Configuration

To ensure consistent execution and rendering performance across different environments, the platform operates using predefined configuration settings.

G. Three-Dimensional Rendering

The final stage of the methodology involves rendering the generated execution traces using React Three Fiber and Three.js. Each supported data structure is implemented as a dedicated visualization component capable of displaying runtime changes dynamically.

TABLE II
SYSTEM CONFIGURATION PARAMETERS

Parameter	Configuration
Execution Timeout	5 s
Maximum Loop Iterations	10,000
Programming Language	JavaScript
Rendering Framework	React Three Fiber
3D Graphics Engine	Three.js
Frontend Framework	Next.js
Backend Runtime Environment	Node.js
Trace Data Format	JSON

The rendering engine manages scene creation, camera positioning, lighting, object animations, and user interactions. By presenting algorithm execution in a three-dimensional environment, learners can more easily understand relationships between data elements and observe state transitions that are often difficult to visualize through traditional teaching methods.

The combination of AST analysis, secure execution, trace generation, and interactive rendering forms the foundation of the DSA Logic Visualizer and enables an engaging learning experience for students studying Data Structures and Algorithms.

IV. EMPIRICAL EVALUATION

To assess the effectiveness of the proposed DSA Logic Visualizer, a small-scale academic evaluation was conducted with undergraduate Computer Engineering students. The objective of the study was to determine whether interactive three-dimensional visualization could improve students' understanding of Data Structures and Algorithms when compared with traditional learning methods.

A. Evaluation Setup

The evaluation involved 40 undergraduate students who had completed introductory programming courses and possessed basic knowledge of Data Structures and Algorithms. The participants were divided into two groups of equal size.

- **Control Group (20 Students):** Learned algorithms using conventional resources such as lecture notes, classroom explanations, textbook diagrams, and static examples.
- **Experimental Group (20 Students):** Used the DSA Logic Visualizer to study the same algorithms through interactive execution tracing and three-dimensional visualizations.

Both groups were given learning tasks based on commonly taught DSA topics including:

- Bubble Sort
- Binary Search
- Linked List Reversal
- Binary Tree Insertion
- Breadth First Search (BFS)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

After the learning session, students were asked to solve conceptual questions, identify logical errors in sample programs, and provide feedback regarding their learning experience.

B. Comprehension Time Analysis

One of the primary evaluation metrics was the time required by students to understand the working of a given algorithm. The average comprehension time was measured from the beginning of the task until the participant demonstrated a correct understanding of the algorithm's execution process.

TABLE III
AVERAGE ALGORITHM COMPREHENSION TIME

Group	Average Time (Minutes)
Control Group	14.2
Experimental Group	8.4

The results indicate that students using the DSA Logic Visualizer required less time to understand algorithm execution. The experimental group showed an improvement of approximately 40.8% compared to students who relied solely on traditional learning resources.

The reduction in comprehension time suggests that visual representation of runtime behavior helps learners understand algorithmic concepts more quickly than static explanations alone.

C. Error Identification Performance

Students were provided with code samples containing intentional logical errors and were asked to identify the source of the problem.

TABLE IV
ERROR IDENTIFICATION RATE

Group	Error Detection Accuracy
Control Group	61%
Experimental Group	82%

Students using the visualization platform demonstrated a noticeably higher ability to identify logical mistakes. The visual representation of algorithm execution allowed learners to observe incorrect state transitions and trace the source of errors more effectively.

An improvement of approximately 21 percentage points was observed between the two groups.

D. Student Feedback Analysis

After completing the study, participants from the experimental group were asked to evaluate their experience using a five-point Likert scale.

Most participants reported that the ability to observe algorithm execution step by step made learning more engaging and reduced the effort required to understand complex concepts. Students particularly appreciated the playback controls and the visual representation of trees, graphs, and linked lists.

TABLE V
STUDENT FEEDBACK SUMMARY

Evaluation Parameter	Average Rating (Out of 5)
Ease of Understanding	4.4
Visual Clarity	4.3
User Experience	4.2
Overall Satisfaction	4.3

E. Observations

The evaluation revealed several important observations:

1) Interactive visualization reduced the amount of time required to understand algorithm execution.
2) Students were able to identify logical errors more effectively when execution states were visually represented.
3) The platform encouraged active learning by allowing users to experiment with code and immediately observe the results.
4) Three-dimensional visualization increased learner engagement and improved conceptual clarity for structures involving hierarchical or interconnected relationships.
5) Students expressed a strong preference for combining traditional instruction with visualization-based learning tools.

F. Discussion

Although the evaluation was conducted on a limited sample size, the results indicate that integrating code execution tracing with interactive visualization can positively influence DSA learning outcomes. The combination of automated trace generation, playback controls, and three-dimensional rendering provides learners with a clearer understanding of algorithm behavior than traditional static approaches.

The findings suggest that visualization-based educational tools can serve as effective supplementary resources for teaching Data Structures and Algorithms, particularly for beginners who often struggle to understand dynamic program execution.

V. CONCLUSION

This paper presented the DSA Logic Visualizer, a web-based platform developed to improve the learning and understanding of Data Structures and Algorithms through interactive three-dimensional visualization. The system combines Abstract Syntax Tree (AST)-based trace generation, sandboxed code execution, and real-time rendering to provide a practical environment in which users can observe the behavior of algorithms as they execute. Unlike traditional visualization tools that focus on predefined examples, the proposed platform allows learners to visualize the execution of their own JavaScript programs, making the learning process more interactive and application-oriented.

The architecture integrates a Next.js frontend, a Node.js backend, and a React Three Fiber-based visualization engine to create a complete workflow from code submission to animated execution playback. By automatically generating execution

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

traces through AST transformation, the platform eliminates the need for manual instrumentation and enables visualization of various data structures including Arrays, Linked Lists, Trees, Graphs, Stacks, and Queues, along with common Sorting and Searching Algorithms.

The empirical evaluation demonstrated that the platform can positively support the learning process. Students using the visualizer required less time to understand algorithm behavior and showed better performance in identifying logical errors compared with conventional learning approaches. Feedback collected during the study also indicated that interactive visualization improved engagement and helped learners develop a clearer understanding of runtime operations and state transitions.

The results suggest that combining code execution tracing with interactive visualization can help bridge the gap between theoretical concepts and practical implementation in DSA education. The modular architecture of the system also provides flexibility for future enhancements and additional educational features.

Future work will focus on extending support to multiple programming languages such as Python and Java, improving visualization capabilities for advanced algorithms, and introducing collaborative learning features that allow multiple users to interact with visualizations simultaneously. Additional research can also be conducted on larger student groups to further evaluate the long-term educational impact of visualization-based learning environments.

In summary, the DSA Logic Visualizer demonstrates how modern web technologies, automated program analysis, and interactive graphics can be combined to create an effective educational tool for teaching Data Structures and Algorithms. The platform offers a practical approach to making algorithm learning more engaging, understandable, and accessible for students.

REFERENCES

[1] S. Halim, F. Halim, and R. Yap, "VisuAlgo: Visualising Data Structures and Algorithms through Animation," National University of Singapore, 2024. [Online]. Available: https://visualgo.net

[2] P. J. Guo, "Python Tutor: Visualizing Program Execution to Improve Programming Education," in *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE)*, New York, NY, USA, pp. 579–584.

[3] D. Galles, "Data Structure Visualizations," Department of Computer Science, University of San Francisco, 2024. [Online]. Available: https://www.cs.usfca.edu/~galles/visualization/

[4] M. Bostock, "Three.js and Interactive Web-Based Visualization Techniques," *IEEE Computer Graphics and Applications*, vol. 44, no. 2, pp. 72–81, 2024.

[5] R. Ferdous, S. Islam, and M. Rahman, "Interactive Learning of Data Structures Using Visualization Techniques," *International Journal of Computer Science Education*, vol. 18, no. 3, pp. 145–158, 2023.

[6] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A Meta-Study of Algorithm Visualization Effectiveness," *Journal of Visual Languages and Computing*, vol. 13, no. 3, pp. 259–290.

[7] A. Sharma and P. Gupta, "AI-Assisted Data Structure Visualization for Computer Science Education," *International Journal of Engineering Research and Technology*, vol. 15, no. 2, pp. 55–63, 2026.

[8] J. Resig and B. Bae, *Secrets of the JavaScript Ninja*, 3rd ed. Shelter Island, NY, USA: Manning Publications, 2024.

[9] B. McKenna and J. Donovan, "Secure Sandbox Architectures for Web-Based Code Execution Platforms," *IEEE Access*, vol. 12, pp. 44512–44525, 2024.

[10] Meta Open Source, "React Documentation," 2025. [Online]. Available: https://react.dev

[11] Vercel, "Next.js Documentation," 2025. [Online]. Available: https://nextjs.org/docs

[12] Babel Team, "Babel Parser and AST Transformation Documentation," 2025. [Online]. Available: https://babeljs.io/docs

[13] P. Hlubuček and M. Kováč, "Visualization-Based Learning Approaches in Computer Science Education," *Education and Information Technologies*, vol. 30, no. 1, pp. 411–429, 2025.

[14] Y. Zhang and H. Liu, "Enhancing Algorithm Comprehension through Interactive Visualization Systems," *Computers & Education: Artificial Intelligence*, vol. 6, Article 100214, 2025.

[15] R. Patel, S. Kumar, and A. Mehta, "Three-Dimensional Educational Visualization for STEM Learning Environments," *IEEE Transactions on Learning Technologies*, vol. 18, no. 1, pp. 31–44, 2025.